Terminal I/O Functions

This chapter covers the following topics:

- Concepts
- Programming Considerations
- Terminal Mapping
- Advanced Facilities
- LU6.2 Transaction Programs
- Syntax
- Extended Graphics Support
- Examples

Concepts

During processing, an online application program is connected to a logical unit (LU). The LU can represent a terminal, either directly or indirectly, through another system (ACCESS, CICS Transaction Routing), or another application program (APPC sessions). The application program communicates with the LU by using Com-plete's *Terminal I/O* Functions. For asynchronous (non interactive) communication with terminals or network printers refer to the*Message Switching*, *Printout Spooling and Terminal Paging* sections in this documentation.

Communication with logical units is governed by conventions (protocols) that apply to each type of logical unit. The Terminal I/O Functions handle the corresponding protocols for the supported LU types in order to free the application program from controlling the sessions.

An application program communicates with an LU by using control commands contained in the data stream that control the processing and formatting of data. Two ways of handling terminal control are available for applications:

- Device-Dependent I/O
- Device-Independent I/O

Device-Dependent I/O

Device-Dependent I/O functions require the application program to to provide and manipulate all terminal control characters for both input and output operations within the data stream itself. These functions must be used*only* when a program is always executed from the same device type or when the program itself is able to recognize the device type from which it was started. Device-dependent I/O functions are *READS* and WRTS.

Device-Independent I/O

The Device-Independent I/O functions automatically supply the necessary control characters and are useful when an application program can be called from different LU types. Output formatting can be accomplished by inserting special characters in the output data stream. On input Com-plete removes all control characters inserted by the terminal before the data is passed to the application. The *READ, WRT* and *WRTT* functions provide device-independent I/O support.

Terminal Mapping

The functions above require data formats to be defined internally - in the application program. Every change in a field format or attribute requires modifications to the program. Input and output data formats can also be defined externally to the application program, in a separate module called*map*. The *READM* and *WRTM* functions use a map as basis of reference for field layout and attributes (terminal control characters). An application can optionally override the attributes defined in the map. Layouts and attributes can be changed without modifications to the program.

Notes:

1. Device dependent and independent I/O functions work identically on LU6.2 sessions since the LU6.2 data stream does not contain control characters. Any character in the stream is treated as data.
2. Do not use Terminal Mapping functions in LU6.2 sessions.
3. Although output with option Done works similar to conversational, it is not possible to read data since the program is terminated without getting control.
4. Output with Return option followed by EOJ will terminate the program but the last record will not be readable by the operator the program is terminated just after displaying the message.
5. For LU6.2 sessions a the Done option works similar to Return + EOJ since there is no way to get a reply from the partner and the last record can be read normally at the other side.
6. The results of the carriage return character can be affected by the*CR* option in the *TIB* macro. Refer to the*TIBTAB*chapter in Com-plete System Programmers documentation for more information.
7. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.
8. Delimiter lists cannot be used with the reread option
9. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Programming Considerations

Program Logic

The Com-plete API makes handling of different protocols almost transparent to the application programmer. It automatically inforces most protocol rules in order to avoid runtime errors so the program design must take into account only the application requirements. It is important, however, to understand

the internal processing logic and restrictions of the used functions in order to get the desired results:

- Input functions do not cause any physical I/O to/from the partner LU; the data was already received either at program startup or as a result of a previous output function. The previously received data is just transfered from Com-plete's buffers to the application program's buffers. A *location counter,* updated for every input request, is maintained to determine how much data from the input buffer was already transferred to the application. The input buffer is freed only when the application received all input data.

- Output functions cause data/control information to be sent to the partner LU. The application is ROLLed OUT and gets control only after completion of the request. Conversational requests are put on the Ready To Run queue only after the reply data is received.

- Input requests may be issued only after conversational output requests, except for the first input after startup that returns the program name and initial data.

- Com-plete's buffer contents can be reread one or more times by specifying the *reread* option *(*suffix *R*). The reread option will prevent updates to the location counter, so the next request will transfer data starting at the same buffer location. Terminal dependent rereads must precede any terminal independent request because the latter will translate all remaining data to device-independent format.

Output Options

All Output requests (except READB) must specify a suffix that indicates the processing logic for the request. The suffix must be either:

R	Return. The application program's thread is ROLLed OUT, the data is sent to the terminal or partner LU but Com-plete keeps the right to send (no CD is sent). The application program is placed in the ready to run queue just after output completion. LU6.2 transaction programs remain in SEND State.
C	Conversational. The application program is ROLLed OUT, Com-plete sends the data and CD to the terminal or partner LU that now can send data back to Com-plete. When the reply is received the application program is put in the ready to run queue and can now issue a READ to retrieve the received data. LU6.2 application programs are now in RECEIVE state.
D	Done. This option works similar to a conversational output followed by EOJ. The difference is that the application does not get control - it is terminated normally after receipt of the operator reply. For LU6.2 sessions the conversation is terminated (CEB), the user logged off and the TIB is removed from the TIBTAB.

Notes:

1. Although output with option Done works similar to conversational, it is not possible to read data since the program is terminated without getting control.
2. Output with Return option followed by EOJ will terminate the program but the last record will not be readable by the operator the program is terminated just after displaying the message.

3. For LU6.2 sessions a the Done option works similar to Return + EOJ since there is no way to get a reply from the partner and the last record can be read normally at the other side.
4. The results of the carriage return character can be affected by the*CR* option in the *TIB* macro. Refer to the*TIBTAB*chapter in Com-plete System Programmers documentation for more information.
5. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.
6. Delimiter lists cannot be used with the reread option
7. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

3270 Terminal I/O Handling

Device dependent I/O

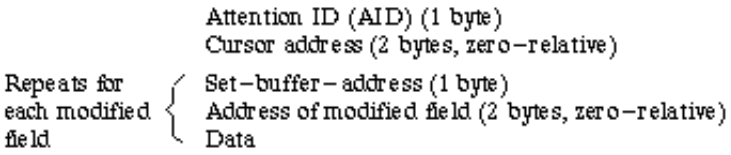
When using device dependent I/O functions the application program must handle the whole data stream. All control characters the device places in the data stream are passed to the program on input. The program must also provide all control characters on output. Errors in control characters may cause unpredictable results.

Com-plete provides special facilities and handling for programmers using device-dependent I/O with IBM 327x compatible terminals.

For ease of use, all 3270 buffer addresses are referred to in the form of 2-Byte binary items relative to zero (16-bit addressing). Thus, row 1 column 1 is x'0000', row 2 collumn 3 is x'0052' (on 3270 model 2 terminals), etc... This holds true for both input and output. Com-plete translates all 12-bit row-column addresses into binary buffer offsets.

For *terminal dependent output* to 327x devices, the 1st output character is taken from the Write Cotrol Character (WCC). The WCC will not appear on the screen.

Only modified data fields from 3270 screens are read. On an initial input of a screen, the following data is presented to the application program:



Field data is variable in length so the most convenient way to process it (without using maps) is probably using a device dependent input (READS) with specifying X'11' (SBA) as delimiter. This enables the program to determine the exact data on a field by field basis. Refer to section *Delimiter Lists*in this chapter for more details.

Device independent I/O

Application programs using device independent input will receive no control characters - all characters will be removed before data is transfered to the application. *Backspace* processing is performed for devices that transfer a backspace character (TTY devices).

For device independent output Com-plete supplies automatically all control characters required by the device in use (at runtime). Data is written/displayed beginning at the upper left-most margin of the page/left-hand corner of the screen using the maximum line length for the device. Outputs longer than 1 line continue on the next available lines. Further output formatting can be accomplished by using embedded special characters:

Character	Meaning	Action
X’00’	Null Character	No formatting action on most devices. Treated as space on some devices.
X’05’	Horizontal Tab	Translated into appropriate code for hardcopy terminal devices. Padding characters may be Required. Some devices ignore this code in the data stream.
X’0C’	Form Feed	Force a "Skip to Channel 1" condition. Ignored if the device does not support form feed.
X’0D’	Carriage Return	Carriage return without line feed.Treated as "new line" if the device does not support the carriage return character.
X’15’	New Line	Carriage return with line feed.
X’16’	Backspace Character	Overlays the previous character written. Accepted by most terminal devices.
X’25’	Line Feed	Causes a space down condition without carriage return.
X’40’	Space	Embedded spaces can be used to affect output formatting.

Notes:

1. The results of the carriage return character can be affected by the*CR* option in the *TIB* macro. Refer to the*TIBTAB*chapter in Com-plete System Programmers documentation for more information.
2. For each *New Line* placed consecutively in the output buffer, a shift for the next new line will be forced.
3. Delimiter lists cannot be used with the reread option
4. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Delimiter Lists

An Application Program may optionally specify a delimiter list - one or more delimiter characters that are expected to be in the input data - instead of an amount of data to be read. Data is transferred up to (but not including) the delimiter.

The delimiter list specified in the DLIST argument is a working storage area in the application program. The format and contents of this area are illustrated in the following table:

Location	Length	Format	Contents
0	2	Binary	Number of Delimiters in this list.
0	2	Binary	Number of Characters returned. Must be initialized to zeros
4	2	Binary	Relative number of found delimiter in list.
6	2	Binary	Delimiters to be used

Notes:

1. Delimiter lists cannot be used with the reread option
2. The Com-plete distribution source library contains sample delimiter lists for COBOL (COBDLST), PL/I (PL1DLST) and Assembler Language (CCDLM) that can be copied into the application program.

Example

Assuming that the application sets the number of delimiters to 2, the delimiter list contains a comma and a period, and the data read by Com-plete that resides in the Com-plete buffer was:

```
JONES,JAMES ALFRED,719 HIGH ROAD.
```

the data returned to the application program by issuing a series of READ functions with the delimiter list option specified would be:

Read Issued	Data Read	Number Returned	Delimiter Found
First read:	JONES	5	1
Second read:	JAMES ALFRED	12	1
Third read:	719 HIGH ROAD	13	2
Fourth read:	-	0	0

Terminal Mapping

Mapping transforms data by using a table called a map. A map defines both the output and input characteristics of a given terminal screen. The map is defined and stored external to the program as a separate load module. In the process of mapping, data fields in the application program and fields in the map are correlated with data fields on the terminal’s screen. The map uses two types of fields, constant and variable. A "mapped terminal write" transfers constant fields from the map and variable fields from the application program to the terminal. A "mapped terminal read" transfers those variable fields that have been modified at the terminal to the application program. On output, the application program can override various map characteristics such as field atributes, cursor location, sound alarm, etc. Modifications to field attributes are done in reference to field names. Mapping optionally informs the application program, by field name, of the fields read and the fields incorrectly entered at the terminal.

A map can be created using the UMAP utility (see the Com-plete Utilities documentation) or assembled and linked using the mapping macros(see the section *Map Creation Using the Macros* later in this chapter).

After the map has been created, the application program can perform terminal I/O using the map by specifying:

1. Mapping Request Control Blocks (MRCBs)
2. The location of the desired data within the application program's working storage area (optional)
3. The location of a Field Control Table (FCT) within the application program which contains field names and dynamic modification of field display characteristics (optional)

The application program consideration(s) for using a map and the process of creating a map are discussed in the remainder of this section. Further information on the use of the UMAP utility program can be found in the Com-plete Utilities documentation.

Map Contents

A map contains two types of data: global data and field data. Some of the information contained within the map may apply only to one type of 3270 (that is, the extended attributes of the 3279 graphics terminals are ignored for non-graphics 3270 terminals of the same size).

Global Data

Global information pertains to how the mapping system will deal with the map as a whole. Global data includes:

- The map name - used to verify that the correct map is being used
- The device code - used to signify the device type for which the map was designed
- Terminal Control Codes (TCCs) - used to specify control options to be used by the mapping system when using the map to write or read data such as Erase-the-Screen
- Global extended 3279 graphics attributes

The TCCs within a map can be overridden at execution time by specifying overriding TCCs within a field of the MRCB. The TCCs are defined inTerminal Control Codes. The extended graphics attributes can be overridden by field in the map or, at execution time, by field in the FCT.

Field Data

Field information pertains to how the mapping system will deal with a specific field. A field within a map can be defined as constant or variable.

A constant field is a field whose fixed text resides within the map. The application program cannot vary the text sent to the terminal or receive the text received from the terminal; however, the application program can modify the display characteristics of constant fields. Constant fields are used for displaying text whose contents are independent of the programs executing (that is, titles of a screen and field prompts).

A variable field is a field whose data resides within the application. Variable field data is moved by a mapped write (WRTMC, WRTM) call from the application storage to the terminal screen. If the field is defined as unprotected (that is, the data can be modified), the data entered on the screen can be returned to the application program with a mapped read (READM) call.

Both constant and variable fields share the following characteristics:

- An optional 6-byte field name;
- A screen location and field length;
- Field Description Codes (FDCs). The FDCs define the display and usage characteristics of a field. At execution time, the FDCs can be altered by specifying an override in the Field Control Table. FDCs describe characteristics such as high or normal display intensity, protected or unprotected. display or non-display, and required or Optional.
  Some FDCs are not valid for constant fields (e.g., unprotected, required, etc.). SeeField Descriptor Codes for the definition of the FDCs;
- Color code for 3279 graphics terminals;
- Symbol set IDs for GDDM symbol set modules.

  Variable fields have the following additional characteristics:

- Data buffer offset. The location of the data to be extracted for the write and the location for returning data from the terminal are determined by adding the data buffer offset to the address passed as the data buffer argument;
- Data types. Variable fields can be alphanumeric, zoned decimal, packed decimal, binary fullword, or binary halfword;
- Number of decimal places. The number of decimal places is used to display packed decimal, binary fullword, or binary halfword fields;
- Internal length. The internal length of packed decimal fields must be defined.

Map Names

The map is stored into the load library under a six-character name. The first four characters are the same as those specified in the name field of the UMAP menu or the MAPSTART macro statement. The last two characters are taken from the device code of the terminal for which the map was designed. UMAP displays this device code after the name field or it is taken from the DEVCODE argument of the MAPSTART macro.

For those applications/installations with performance problems due to excessive reloading of specific maps, the map can be placed into the resident list of Com-plete. Mapping will search first the thread, second the resident area of Com-plete, and finally the load library chain.

For maps accessed via the resident area of Com-plete that require scaling (see the following section), mapping will copy the map to the thread temporarily, scale the map, use the map, and free the thread storage.

Device-Specific Mapping and Scaled Mapping

This section describes the use of the device code and choice of using maps in a device-specific manner or in a scaled manner. Terminal Device Type Codes gives the device codes associated with each screen size of 3270 terminals.

Assume that your application uses only one screen. If this application program is required to operate from only one specific terminal device type, the 24-line 80-column F2, then one map is Required.
You could name this map XXXXF2 and set the MRCB as follows:

```
Map name is 'XXXX'.
Version is 'B'.
Filler is three spaces.
Mapvers is space.                    (See Mrcb Exception Codes for MRCB format)
```

Mapping concatenates the XXXX to the device type, resulting in XXXXF2. If the application is executed from a different device type (for example, an F5), the application needs an XXXXF5 map with the same field names and characteristics. These maps are device-specific; a unique map exists for each terminal type, as necessary.

In some situations, having a unique map for each device type allows the application to display more data on larger screens and less data on smaller screens.

Some applications take no advantage of the differences in 3270 screen sizes. If the screen layout of the map fits within the dimensions of another device type, the application can request mapping to use map XXXXF2 but scale the map to fit on the current device. To request scaling, set the MRCB as follows:

```
Map name is six characters, i.e., 'XXXXF2'.
Filler is two spaces.
Mapvers is 'B'.
```

Note that scaling relocates the start of each field. Users of scaling should not use fields that wrap off of the right side of the screen and back on to the left.

Program Concepts

The application program *must* provide an MRCB in the working storage area of the application program. The MRCB contains the name of the map.

The program can optionally provide the FCT, if it is necessary to dynamically modify the display characteristics of specific fields or to receive more detailed information about input fields.

Since one map defines both the output and input handling, a typical application program performs a write-mapped call followed by a read-mapped call using the same map and same MRCB.

When a mapped read or write call is executed, mapping determines the name of the map by concatenating the MRCB map name field with the terminal device code. Mapping determines the location of the map and loads the map into thread storage, if necessary.

The manner in which individual fields are processed is determined by information passed in the MRCB and the optional buffer and FCT parameters. The MRCB is used to indicate that individual fields in the map are to have their processing characteristics controlled by the application program. When this indication is given, the CALL statement normally supplies the FCT parameters in which the overriding characteristics of the desired fields are specified.

The WRITE-OPTION of the MRCB allows the application program to indicate which of the following methods mapping should use:

- Write all fields defined in the map, optionally overriding the display characteristics for those fields entered in the FCT;
- Write only those fields specified in the FCT, optionally overriding the display characteristics.

  The READ-OPTION of the MRCB allows the application program to choose among the following:

- Read all readable fields;
- Read only those fields specified in the FCT.

Note that a mapped write with no FCT and no buffer can be used to write only the constant fields.

The MRCB, FCT, and CALL statement conventions are discussed in detail in the following sections.

MRCB

The MRCB is a working storage area defined within the application program. It contains the name of a map, terminal control information, and mapping field control information. Note that an application program may contain more than one MRCB, but *only* one MRCB is Required.

Users are provided MRCB copy code for COBOL, PL/I, and Assembler. The format of the MRCB, along with a description of each of its fields, is found inMapping Request Control Block (MRCB).

In Assembler language, a map can be coded in line in the program and located immediately behind the MRCB. The program can then be assembled with the map located directly in line with the program, thus saving a load operation for the map. If this technique is to be used, the MAP-CONCAT fields in the MRCB must be initialized to a C.

Another method for specifying that the application has the map in storage is to set the MAP-CONCAT field in the MRCB to A and the MAP-ADDRESS field of the MRCB to the location of the map.

There is no logical restriction on how many maps a program can use. From a performance standpoint however, if multiple maps are to be used, it may be desirable to make some or all of them resident in the thread region of the application program. The MAP-COUNT field of the MRCB is used to request this option. This value literally creates a queue of thread resident maps. The number of entries in the queue is equal to the number specified in MAP-COUNT. If more maps are referenced than the queue can accommodate, the queue of maps is treated as a "first-in-first-out" queue. A map-count of zero signifies that the map should be used and then deleted.

Since the MRCB is used to pass control information back and forth between the application program and Com-plete, some of the MRCB fields must be set by the application program and some by Com-plete. Consequently, the MRCB is required for all mapping requests. The default value for all MRCB fields is spaces, with the exception of the MAPNAME and VERSION fields.

FCT

The FCT is defined within the application program in the working storage section. Its function is to enable the application program to change the display characteristics of individual fields and/or to receive more detailed information about each field.

The FCT, if defined, must consist of one FCT entry per field to be individually processed. Each entry is referred to as an FCTE.

If the FCT is not a parameter in the CALL statement, each field in the map is assumed eligible for writing, and all unprotected fields in the map are eligible for reading.

Each FCTE must be defined in one of three formats:

- Short format of 6 bytes, or:
- Long format of 10 bytes, or:
- Extended format of 13 bytes.

Note that new applications must be coded using the extended format. The short and long forms have been retained only for compatability with existing programs.

The short form contains only field names. The long form contains the field name, an input flag, and a Field Descriptor Code (FDC) override field. The extended form also contains the override color and override symbol set ID for 3279 graphics. The format used must be indicated in the MRCB.

The format for each type of FCTE is defined in Field Control Table (FCT).

Buffer Area

The buffer area(s), or record area, into which data is to be placed during a read operation and from which data is to be obtained during a write operation must be defined within the application program.

When an application program uses an existing file record definition, the programmer can specify the data offset during map creation. If an existing record format is not being used, use the UMAP edit copy code function to create a buffer.

Output Field

WRTM is the mapping function used for writing information or data to the terminal. This function is described in detail later in this chapter in the section entitled "WRTM".

Output processing involves global/field information from the map and dynamic overrides from the application program. Terminal Control Codes (TCCs) are stored in the map when the map is created, and they are overridden by specifying TCC-OVERRIDEs in the MRCB.

The TCC codes allow for the following sets of control:

E/N	E: Erase unprotected fields prior to the write. N: Specifies that these fields will not be erased. An application program can wish to rewrite only specific fields and have the remaining unprotected fields erased or not.
A/Q	A: Sound audible alarm at terminal. Q: Alarm is not sounded.
P/S	P: Start printer. S: Printer is not started.
K/M	K: Turn off the terminal’s modified data tags. The modified data tags indicate that an unprotected field has been modified.
R/L	R: Unlock the keyboard. L: Lock the keyboard.

TCC codes affect the erasure and reformatting of the constant fields. In the following discussion, reference is made to options for which Com-plete determines whether an action is necessary. If these options are selected (B and F), Com-plete determines if the same application program and map was used for the previous write to the terminal and that no message switching, paging, terminal clear operation or program fetch was done. This determination should be sufficient to keep the screen correctly displayed with a minimum of rewriting of constant fields. Application programs that involve overlaying of mapped screens may need to force no erase or force the formatting of constant fields.

B/W	B specifies that Com-plete should determine whether the screen requires erasing before the write. W specifies that the screen should not be erased.
C/D/F	F specifies that Com-plete should determine if constant fields need to be rewritten. Specify D to force mapping to do no rewriting of the constant fields, or C to always force writing of the constant fields.

With these functions in mind, the application programmer can use the WRTM function to write the following information to the terminal:

- Write only those fields defined as constants in the map.

  This option can be forced by not passing the buffer area or the FCT when executing the WRTM function.

- Write all the fields defined in the map, (optionally) overriding the display characteristics for those fields entered in the FCT.

  This option can be forced by entering a space or an A in the WRITE-OPTION field of the MRCB, passing the buffer area in the WRTM function, and (optionally) passing the FCT argument.

- Write only those fields specified in the FCT, (optionally) overriding the display characteristics.

  This option is forced by entering an O in the WRITE-OPTION field of the MRCB and passing both the buffer area argument and the FCT argument.

Output validation is performed for data being written to the terminal to determine whether the program data area field contains data that can be properly edited and moved to the map field entry. Specific output validation performed is summarized below.

Alphanumeric and zoned decimal fields:

- Transferred from the program data area without validation.

Packed and binary fields:

- If the field contains invalid data, the program is terminated abnormally.
- Leading zeros are suppressed.
- If indicated in the map, a decimal point is edited into the display.

- A "-" immediately precedes either the high order digit or the decimal point, if the field is negative.
- For zero value fields, a single zero or the decimal point and all decimal places are displayed.
- A numeric attribute is forced, unless the field is specified as skip or protected (FDCs of S or P).
- If a value is too large to fit in the map display field, the display field is filled with asterisks.

Input Field Processing

READM is the mapping function used for processing input data from mapping requests. This function is described in detail in this section.

Input fields are processed according to location (that is, row and column) on the screen; therefore, the map used to read them should correspond exactly to the formatted screen. This can be easily accomplished by using the same map to both read and write the screen.

Input validation is automatically performed for data being read from the terminal to determine whether the input data can be properly edited and moved to the program field areas. The specific input validation performed is summarized below.

Alphanumeric fields:

- Validation is performed for length only.
- If more data is entered than the program-defined field can accommodate, an overflow exception condition will be posted.
- If not enough data is entered to fill the field, the data will be left-justified and space-filled.

Zoned decimal fields:

- Validation is for characters 0 - 9.
- Data can contain leading and/or trailing blanks.
- Data in the program data area will be right-justified and zero-filled.
- Possible exceptional conditions posted are INVALID DATA and OVERFLOW.

Packed fields:

- Input can have leading and/or trailing blanks.
- If negative, the first digit or the decimal point must be preceded by a "-".
- Decimal point placement is indicated by a period.
- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data field area.

- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

Binary fullword fields:

- Negative numbers are stored in two’s complement form.
- Input can have leading and/or trailing blanks.
- If negative, the first digit or the decimal point must be preceded by a "-".
- Decimal point placement is indicated by a period.
- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data area.
- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

Binary halfword fields:

- Negative numbers are stored in two’s complement form.
- Input can have leading or trailing blanks.
- If negative, the first digit or the decimal point must be preceded by a "-".
- Decimal point placement is indicated by a period.
- Data must be numeric, except as indicated above. Data is aligned, converted, and stored in the program data area.
- Possible input exceptions posted are INVALID DATA, OVERFLOW, and UNDERFLOW.

  The MRCB can contain a variable length feedback area. If so, this area is used to indicate input errors from the terminal. Data entered that conflicts with the field definition for the mapping field in which it is entered is not returned in the buffer area. Instead, the name of the mapping field, followed by an exception code, is listed in the feedback area. The MRCB feedback area exception codes are defined in Mrcb Exception Codes.

To illustrate the use of exception codes in the MRCB feedback area, consider the following example where the underscored data was entered at a terminal.

```
Gross Pay .01 SSN A00000000
```

If the field GPAY was defined as having less than three decimal places in the map, and if the field SSN was defined as a numeric field, then the MRCB feedback area would contain:

```
GPAY  UF,SSN  NN,
```

Note that each field entered in error is placed in the MRCB feedback area. The format of each entry in the feedback area is illustrated in the following figure. The number of fields in the feedback area is indicated by the MRCB's error-fields. Note also that the feedback area is not initialized between reads.

FIELD-NAME	OFFSET	LEN	Description
ERROR-FIELD	0	6	Name of the field in error
FILLER	6	1	Blanks
ERROR-CODE	7	2	Error codes, as described in Mrcb Exception Codes

In addition, three fields within the MRCB are updated. The MRCB's CURSOR-IN field contains the input field name cursor location. The FIELDS-READ and ERROR-FIELDS fields are used to indicate the number of fields returned to the application and the number of errors detected.

If an error is detected while processing a read function, a return code is posted in the return code field of the MRCB and in the*retcode*argument. The data for the field or fields in error is *not*transferred to the program data area.

In addition to the input exceptions posted in the MRCB feedback area, an input indicator is placed in each input flag field in the FCTE, if the long or extended format of the FCTE is being used (unless the FCTE is specified as ignored or protected). The codes returned are listed inField Control Table.

Map Creation Using Macros

Before creating a map, you should design a separate display layout of each map for each terminal device type to be used. Currently, the only device types supported by mapping are 3277 models 1 and 2, 3278 models 1 through 5, and the graphics terminal or compatible devices. These device types are referred to as formattable devices; other device types are non-formattable devices.

After the map layouts have been designed, the macro statements defining the appropriate maps can be written. For example, consider the following display for which a map definition is desired:

```
NAME:    fred schwartz
ADDRESS: 1208 sw street
```

The map definition used to generate this display is:

```
MP01 MAPSTART F2
  MAPF     ,'NAME:'
NAME  MAPF     (1,10),14
  MAPF     (2,1),'ADDRESS:'
ADDR MAPF      ,20
  MAPEND
  END
```

In the above sample map definition, note the entries accompanying the MAPSTART macro statement. The entry MP01 is the name of the map and the entry F2 is a terminal device type designator.

After the map has been defined, it is ready to be assembled and link edited. The assembly of the map should be performed using the ALGN option of the assembler or the results may be unpredictable.

MAPSTART Macro

The MAPSTART macro is used to identify (name) the map, specify the device class code of the terminal(s) on which the map is to be used, and specify optional terminal control information.

The format of the MAPSTART macro is:

```
name    MAPSTART  [devcode]
    [,FDCDEF=]
    [,TCC=]
     [,COLDEF=]
     [,PSDEF=]
     [,TYPEDEF=]
```

All the arguments, except *name*, are Optional.
These arguments are:

name	Required.Default: None.The name of the map. /The name must be exactly four or six alphanumeric characters in length and must begin with an alphabetic character.
devcode	OptionalDefault: F2 The device class code of the terminal for which this map is to be used.The devcode must be one of the following:
	F1 for 3270 model 1 or compatible device (12 lines x 40 columns). F2 for 3270 model 2 or compatible device (24 lines x 80 columns). F3 for 3278 model 3 or compatible device (32 lines x 80 columns). F4 for 3278 model 4 or compatible device (43 lines x 80 columns). F5 for 3278 model 1 or compatible device (12 lines x 80 columns). F6 for 3278 model 5 or compatible - (27 lines x 132 columns).
FDCDEF	Optional.Default: DTO; no extended attributes.The default to be used for the Field Descriptor Code (FDC) argument of the MAPF macro as used in this map definition.The available FDCs are listed in Field Descriptor Codes .
TCC	Optional. Default: BEKQRSW The Terminal Control Codes (TCCs) to be used when performing write commands. See Terminal Control Codes for further details. If any TCCs are specified, at least one of the following pairs of TCC codes must also be specified: AQ, BW, DF, EN, KM, LR, or PS.

COLDEF	Optional. Default: No default color assigned.The default color for the COL argument of the MAPF macros used in this map definition and the background color for the entire screen are: BL RE PI GR TU YE NE or blank, which applies to 3279 graphics terminals only.
PSDEF	The default program symbol set ID for the PS argument of the MAPF macros used in this map definition.
TYPEDEF	Optional. Default: AThe default for the TYPE argument of the MAPF macros used in this map definition:

A	Alphanumeric
F	Fullword
H	Halfword
P	Packed
Z	Zoned decimal

MAPF Macro

The MAPF macro is used to define each field to be displayed, including title fields and fields from the application program.

The specification of row and column locations for display fields must allow for a one-character filler entry that precedes each field in the display. For formattable devices, this field is reserved for the hardware-controlled attribute byte. For non-formattable devices, a blank is inserted in this location. This permits identical displays for both formattable and non-formattable devices.

The format of the MAPF macro is:

```
[name]   MAPF  [(row,column)]
    {,length1|,'constant'}
     [,length2]
     [,repeat]
    [,DECPLAC=]
     [,FDC=]
     [,COL=]
     [,PS=]
     [,OFFSET=]
    [,TYPE=]
     [,ITR=]
```

All the arguments are optional except *length1* and*constant*, between which a choice must be made. Note that the absence of the (row,column) argument must be shown by a comma. Fields must be specified in sequence, by column, within row, and can not overlap.

The arguments are:

name	Optional. Default: The field will be unnamed.The name is used in the FCT for field modification and feedback and in the MRCB feedback area during input field exception processing.The name must begin with a letter and cannot exceed six characters in length.
(row,column)	Optional. Absence of this argument must be specified by a comma.Default: The field is concatenated to the previously-defined field. If this is the first field defined, it is placed in location (1,1). The terminal display location for this field. The first terminal display position is (1,1). If (row,column) is omitted, the field immediately follows the previous field in this display. Note that an apparent space exists because of the attribute byte.
length1	Optional. Default: The length is derived from the length of constant, if entered; otherwise, it must be specified, or an error is generated.The display length for the field. For alphanumeric (type A) and zoned decimal (type Z) fields, it also specifies the data area field length within the application program using this map. This length does not include the filler byte.
constant	Optional. Default: If omitted, length1 must be specified.A character string to be placed in the display field. The display length of the field is determined by the number of characters entered in this argument. The maximum number of characters allowed is 255. The FDC for this field is forced to include S for format table devices.
length2	Optional. Default: Must be specified with packed fields.The data field length, as it exists in the application program for packed (type P) fields. The length is specified in bytes. The field cannot exceed eight bytes in length.
repeat	Optional. Default: 1The number of times, plus 1, that the constant is to appear in the terminal display in the same field. The length1 value for the field is derived by multiplying the length of the constant by the repeat factor.
DECPLAC	Optional. Default: 0 The number of decimal places in this field. This argument can only be specified if the TYPE is P, F, or H for this field. This argument cannot be specified if the constant argument was given.This argument is used to align the decimal point on input fields, and for editing decimal points on output fields. The maximum value is 15.

FDC	Optional. Default: DOTY The Field Descriptor Codes to be associated with this field. More than one FDC can be given. In case of conflict, the last one in the string takes precedence. If the constant argument was given, either S or P is assumed and cannot be overridden; however, any of the other allowable codes may be specified for the class of device for which the map is being used.
COL	The two-character color attribute to be associated with this field on a 3279 graphics terminal. The default is set from the COLDEF argument of the MAPSTART macro. Values: BL//RE//PI//GR//TU//NE or blank
PS	The one-character programmed symbol set ID to be associated with this field on a 3279 graphics terminal. The default is set from the PSDEF argument of the MAPSTART macro.
OFFSET	Optional. Default: The data field in the program working storage area is assumed to be concatenated to the last field specified with a positive offset, whether or not the offset was implied or specified.The number of bytes, either negative or positive, from the beginning of the buffer I/O area to the location of this field within the application program. The numerical value can range from -32768 to +32767. By adding this value to the data area argument passed in the READM or WRTM call, the location of the field in the program can be determined.
TYPE	Optional. Default: "A" or the value specified in the TYPEDEF argument of the MAPSTART macro.The type of field within the program data area.This argument must not be specified if the constant argument is given.
	A Alphanumeric F Binary fullword H Binary halfword P Packed Z Zoned decimal K Kanji
ITR	Optional Default: OFFSpecifies whether input translation is to be performed on this field. This argument can only be specified for alphanumeric-type fields.

MAPEND Macro

The MAPEND macro allows for error detection, end-of-map processing, and the display of information about the map. The MAPEND macro is required.

The format of the MAPEND macro is:

```
MAPEND
```

Advanced Facilities

Structured Fields

Structured fields are used to convey additional control functions and data to or from the display terminal. Write Structured Fields (WSF) is the only 3270 command that can be used to send structured fields from the application to the display. This command may be used only for devices that support extended data stream. Devices that do not support WSF will reject this command with SENSE X'1003'.

Functions that can be accepted by display devices include partition/screen handling, outbound text or data streams and partition read. The display uses and AID (X'88') to indicate inbound structured field functions.

WSF commands can be issued using the*WRTSF*and*WRTSFC*functions. The application program must provide the complete command.*WRTSFC* followed by a device-dependent input function (*READS)*should be used if the command specifies an application-initiated read (Read Partition or Read Buffer).

For Read Buffer the application program can use the special function*READB* instead of *WRTSFC*. READB has no parameters and automatically sends the Read Buffer command.

For more information about WSF refer to the IBM 3270 Information Display System *Data Stream Programmer's Reference (#GA23-0059-1),* and the *3274 Control Unit Description and Programmer's Guide(#GA23-0061-2).*

Periodic Redisplay

Output functions cause the application program to be ROLLed OUT. A time parameter (Default=0) can be specified to tell Com-plete the elapsed time before the application program is put back in the ready-to-run queue. This feature can be used in *non-conversational* output requests to refresh a screen at periodic intervals either with updated data or a constant message. The *TESTAT* function (test for attention interrupt) can immediately follow the timed output function to terminate the output loop. Refer to the*Miscelaneous Functions* chapter in this documentation for a description of*TESTAT*.

Note:
Application programs using this feature should not run on devices that do not support attention interrupts such as CICS Transaction Routing or LU6.2 sessions since there will be no way to interrupt the loop.

Time-out

For conversational output functions the application program is put in the ready-to-run queue only after the reply from the partner or device is received. If the *time* parameter is specified the application program, as in non-conversational functions, is put in the ready-to-run queue after the specified time is elapsed. If the application issues a subsequent input function before Com-plete received the reply no data will be returned to the application program, thus identifying a *time-out*. The application program can then take the necessary actions (backout, terminate, etc.). This feature is helpful to avoid program hangs on LU6.2 sessions or pseudo-terminals when the partner application fails to respond.

LU6.2 Transaction Programs

Com-plete handles LU6.2 Server Transaction Programs (TP) as normal online programs. From the user's point of view they just communicate with the partner transaction program instead of a terminal.

To start a server TP in Com-plete the partner (client) TP must specify the program name (up to 8 characters) in the ALLOCATE verb. Com-plete receives this TPname together with the logon information in the ATTACH FMH-5. The user is then logged on and the TP started (as if *TPname was entered at a display terminal). If the security information is incorrect, an error will be returned to the partner TP.

Server Tps always start in RECEIVE State and must issue a READ function. The 1st READ will return the program name followed by the first Logical Record (Basic Conversation) or Mapped Conversation Record.

Unlike display devices that send 1 screen at a time, LU6.2 TPs may send chains containing more than 1 logical record/mapped conversation record but only one record can be received for each input request. Since input requests may be issued only after conversational output (except for the first input and when reread is used) the TP must issue a conversational output request without data so a subsequent input request can receive the next record. Note that this feature is valid only for LU6.2 sessions.

When in SEND State, several records can be sent in 1 chain by using non-conversational output functions (Write Return). Write Conversational causes a CD (Change Direction) to be sent to the partner and the conversation state is changed to RECEIVE.

Write Done terminates the conversation (implicit DEALLOCATE).

Restrictions

Com-plete V5.1 Communication Functions currently provide limited LU6.2 support since the API - originally designed for terminal I/O - does not provide some parameters and functions required for full LU6.2 support. An extension to the API is planned for future releases.

LU6.2 verbs like ALLOCATE, CONFIRM, CONFIRMD and parameters like Conversation ID (CONVID), Partner LU Name, MODENAME, Conversation Type currently cannot be explicitly specified or received due to API restrictions what leads to some programming limitations:

- LU6.2 Transaction programs can currently run only as SERVER programs. All needed parameters are provided by the client in the ALLOCATE verb.

- Currently there is no way to pass control information - conversation parameters (modename, synclevel, session type), conversation states, what_received - to the TP. The programmer must know almost the exact data flow when designing the application to comply to the LU6.2 protocol. For example, the programmer must know exactly how many records will be contained in each chain and when a CD is expected so the TP can send data to the partner. If the conversation is in RECEIVE state Com-plete will ignore any data specified in the conversational output until all records of the current chain were received.

- Conversational output, when in SEND state, cause Com-plete to send data + CD to the partner, thus entering RECEIVE state. This must currently be (mis) used to force Com-plete to receive the next record of a chain when in RECEIVE state. Any data specified in these dummy requests is ignored. The application program cannot force SEND State (SEND ERROR) when in RECEIVE state.

- A TP may handle only 1 Conversation

Syntax

Device-Independent Input: READ

```
READ[R] (retcode,area,length[,numleft|,numread[,dlist]])
```

Device-Dependent Input: READS

```
READS[R]  (retcode,area,length[,numleft|,numread[,dlist]])
```

Parameters:

R	OptionalReread Option.
retcode	Required.A fullword where Com-plete places the return code on completion of the operation.
area	Required. The buffer area in the working storage area of the application program where Com-plete places the data to be transferred from Com-plete's buffer.
length	Required.A binary halfword containing the number of characters to be transferred from Com-plete's buffer. length must be greater than zero.
numleft	Optional.A binary halfword where Com-plete places the number of characters remaining to be transferred before the READ without reread option was issued.
numread	Optional.A binary halfword where Com-plete places the number of data characters actually transferred from Com-plete's buffer to the application program buffer when a READ with reread option is specified.
dlist	Optional. Not applicable if the reread otion is used.The working storage area of the application program which contains the delimiter list to be used with the READ request. This area must have been previously defined and initialized by the application program.

Return Codes

Application programs should check the return code for one of the following values:

0	The amount of data transferred to the application program is equal to the amount of data in Com-plete's buffer.
4	The amount of data transferred to the application program is less than the amount of data in Com-plete's buffer.
8	The amount of data requested for transfer to the application is larger than the amount of data in Com-plete's buffer. Existing data is transferred, but extra buffer space is not modified.

Abends

An abnormal termination can occur during execution of the READ function. Possible causes include:

- An invalid argument was specified;
- The input area is not in the user area;
- The length specified is negative.

Input Using Map: READM

```
READM[R] (retcode,mrcb,darea [,fct])
```

Parameters:

R	Optional.Reread Option.
retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
mrcb	RequiredThe name of the MRCB as defined in the application program. The MRCB must be defined on a fullword boundary.
darea	Required.The name of the buffer data area in the application program where the input fields are to be placed.
fct	OptionalDefault: None. The name of the FCT in the application program that will be used according to the MRCB READ-OPTION field and the MRCB FCTE-FORMAT field and MRCB-FCTE count.

Return Codes

The return code, placed both in the first argument and in the RETURN-CODE field of the MRCB should always be examined for one of the following values:

0	No input errors were encountered and all required fields were read.
4	At least one error was encountered in the input. The MRCB ERROR-COUNT field contains the number of fields in error and the number of field names with exception codes in the feedback area.
8	There is at least one field in error and no feedback area was specified, or if a feedback area was specified, it is full.
12	The location of a field in the input does not match a field location specified in the map.

Data can be altered in the application program buffer area regardless of the return code value received.

Abends

An abnormal termination may occur during execution of the READM function. Some possible causes are:

- An invalid MRCB was found;
- An invalid area argument was specified;
- An invalid FCT argument was specified;
- The MRCB was not on a fullword boundary.

Device-Independent Output: WRT

```
WRT[T]{C|D|R} (retcode, area,length[,linelen[,time]])
```

WRT*T*x specifies a device-independent output function with*T*ext option. All data written to the terminal is separated into logical words that cannot be partially contained in one line. If the word does not fit completely on the line it is displayed on the next line.

Device-Dependent Output: WRTS

```
WRTS[E]{C|D|R} (retcode,area,length[,linelen[,time]])
```

WRTS*E*x specifies a device-dependent output operation with prior erasure of thet a 3270-screen.

Special Output

WRTSF - Write Structured Fields

```
WRTSF{C|D|R} (retcode,area,length[,time[,plist]])
```

READB - Write "Read Buffer"

```
READB
```

Parameters:

C/D/R	Required.Specifies the processing logic for the request. Refer to section Programming Considerations for more details.
retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
area	Required.A buffer area in the application program containing the data to be written to the terminal.
length	Required.A binary halfword containing the number of characters of data to be written.
linelen	Optional.A binary halfword containing the value of the logical line length to be used for the terminal. The linelen argument cannot be specified for terminal-dependent write requests. Default: If linelen is not specified, or if a linelen of zero is specified, the physical line length of the terminal is used.
time	Optional.A binary halfword containing the number of seconds after which the application program is placed at the bottom of the Com-plete ready-to-run queue to await dispatching. When used with the "return" form of the WRT request, control is returned to the application after the specified length of time has elapsed.When used with the "conversational" form of the WRT request, control is returned to the application when an interrupt occurs at the keyboard, or after the specified length of time has elapsed. The time-out can be identified by the fact that a "read" request returns no data.Default: None. The application program is placed immediately in the ready-to-run queue.

Return Codes

The application program must examine the first parameter after completion of the request for one of the following return code values:

0	The write operation was successful.
4	The write operation was terminated by the terminal user, either by pressing the <CLEAR> key (or its equivalent), or by entering the character string *EOJ. The application program can optionally choose to ignore this circumstance and continue normal execution.
8	The terminal operator has terminated the write operation by entering the character string *CANCEL, or the stack level has been terminated. If a terminal I/O with an option other than DONE is issued after a return code 8 is received, the application program is abnormally terminated
Â	This value is normally reserved to enable the application program to perform logical end-of-job processing.
12	A terminal I/O error has occurred. When return code 12 is received, the application program is abnormally terminated, if another terminal I/O function that does not specify the DONE option is executed.
16	The output created by execution of the WRT function was destroyed at the terminal. Normally, this condition occurs if, while viewing the output, a message was sent to the terminal that destroyed the formatted output, or if the terminal user temporarily suspended this program in order to retrieve another. The application program should reissue the WRT request to force a rewrite of the entire screen. Mapping automatically handles this condition.

Abends

During execution of the WRT function, an abnormal termination may occur. Some possible causes are:

- Too many output lines were requested to be written.
- The area or length arguments were invalid.
- The terminal operator entered a reply of *CANCEL and the application program executed another WRT request other than WRTxD.

Output Using Map: WRTM

```
WRTM{C|D|R} (retcode,mrcb [,darea] [,fct])
```

Parameters

C/D/R	Required. Specifies the processing logic for the request. Refer to section Programming Considerations for more details.
retcode	Required.A fullword where Com-plete places the return code upon completion of the operation.
mrcb	Required.The name of the MRCB as defined in the application program. The MRCB must be defined on a fullword boundary.
darea	Required if the FCT parameter is specified. The name of the buffer data area in the application program where the output fields are obtained during WRTM processing.Default: If omitted, the format and fields from the map are written.
fct	Optional.The name of the FCT in the application program that is used according to the MRCB WRITE-OPTION field and the MRCB FCTE-FORMAT field. This argument need not be specified if all the fields specified in the map are to be written without modifying their display characteristics.

Return Codes

Return codes are placed both in the first parameter and in the MRCB RETURN-CODE field. Possible return codes and their meanings are:

0	Normal return.
4	The operator entered *EOJ or pressed the <CLEAR> key.
8	The operator entered *CANCEL in the first field, or the stack level has been terminated.
12	A terminal hardware error was detected during the WRTM operation.
16	The screen format has been erased. The screen format can be erased by the terminal that has received a priority message, or destroyed by a user who has suspended the application.

For formattable devices, it is assumed that the format is destroyed, if a return code other than zero is passed to the application program. In this situation, the format is automatically rewritten when the next WRTM request is executed, unless the WRITE-OPTION field of the MRCB specifies the letter O (only).

Abends

An abnormal termination may occur during execution of the WRTM function. Possible causes include:

- An invalid MRCB entry was given;
- An invalid area argument was specified;
- An invalid FCT argument was specified;
- The MRCB was not on a fullword boundary.

Extended Graphics Support

Extended 3279 graphics terminals have capabilities not supported by other 3270 models. Mapping support for these devices is implemented so that:

- Maps created for the non-extended will function on the extended models;
- The extended attributes will be ignored for non-extended models;
- Extended capabilities are defined on the basis of global and field definitions (no subfielding capabilities).

The extentions include color attributes, customized symbol sets, and extended highlighting.

These features are available with UMAP's TCC UPDATE function on the global level, and with UMAP's ATTRIBUTE UPDATE function on the field level. For example, you can specify the color of the screen as pink, as well as a different color for each field. As before, these attributes can be overridden by use of the FCT.

Symbol Sets

Extended graphics terminals can be loaded with multiple user-defined symbol sets, which define the shape and color of any screen symbol. For further information on the creation of symbol sets, see the *IBM User's Guide for the Graphical Data Display Manager*.

The symbol sets are stored in either VSAM files or STEPLIB libraries as modules with eight-character names. Rather than specifying an eight-character name, mapping support refers to these symbol sets by a one-character symbol set ID. Applications must have the device loaded with the correct symbol set and symbol set ID.

Loading Symbol Sets

Symbol set modules can be loaded under application program control by using Com-plete's COLINK function and the Com-plete subroutine U2MASS. For the purposes of testing, the UMAP "LOAD PROGRAMMED SYMBOLS" function can be used to load symbol sets with an associated symbol set ID, and UMAP will call U2MASS.

Format

The format for using the COLINK function is:

```
COLINK (retcode,subroutine-name,entry1)
```

<table>
<tr><td>retcode</td><td>Required.A fullword where Com-plete places the return code upon completion of the operation.</td></tr>
<tr><td>sub-routine-name</td><td>An eight-character field with value "U2MASS".</td></tr>
<tr><td>entry1</td><td>An eleven-byte structure for each symbol set, to be:
<table>
<tr><td>Offset</td><td>Length</td><td>Contents</td></tr>
<tr><td>0</td><td>8</td><td>symbol set module name</td></tr>
<tr><td>8</td><td>1</td><td>symbol set ID</td></tr>
<tr><td>9</td><td>1</td><td>storage plane requested</td></tr>
<tr><td>10</td><td>1</td><td>storage plane assigned</td></tr>
</table>
where:
symbol set name
Specifies the name of the GDDM-generated symbol set name module.
symbol set ID
Is a one-character ID by which mapping refers to the symbol set.
storage plane
Is a one-character storage plane name, requested as defined in 3270 component description, to be used by the module.
storage plane
Is assigned by U2MASS. used</td></tr>
</table>

Examples

Example 1 - Terminal-Independent I/O

This sample program demonstrates the use of terminal independent READ (also with reread option) and WRT using C, D and R options and also the time parameter. See the program comments for usage details.

```
COPY  CCGLOBS
*
*         REGISTERS ON ENTRY:
*               R2 = A(COMREG)
*               RD = A(CALLER'S SAVE AREA)
*               RE = RETURN ADDRESS
*               RF = ENTRY POINT
*
SAMP1     CSECT
          USING SAMP1,RC
          STM   RE,RC,12(RD)
          LR    RC,RF                    LOAD ENTRY POINT
          ST    RD,SAVE+4
          LR    R3,R1                    SAVE A(PARMS)
          LA    R1,SAVE
          ST    R1,8(RD)
          LR    RD,R1
          LA    R0,IPTAREA
          MVC   OUTAREA,BLANKS
```

```
         MVC   PROGNAME,BLANKS
         LA    R6,WSTABLE
         LA    R5,LASTENT
*
*        GET TOTAL LENGTH ENTERED
*
         CM$CALL READR,(RETCODE,IPTAREA,WRLEN,NUMREAD)
         LH    R1,NUMREAD
         CVD   R1,DWRD
         UNPK  TLEN(2),DWRD+6(2)
         OI    TLEN+1,X'F0'
*
*        SKIP PROGRAM NAME
*        *SAMP1 = 6 CHARACTERS + 1 BLANK = 7
*
         CM$CALL READ,(RETCODE,PROGNAME,PRGLEN)
         CLI   RETCODE+3,4             AMOUNT OF DATA TRANSFERED IS...
         BH    CANCEL                  LESS THAN REQUESTED. ERROR
         BL    DISPLY                  = REQUESTED ==> PROGNAME ONLY
*
*        THE TEST BELOW CAN BE USED INSTEAD OF THE PRECEDING:
*
*        CLI   NUMREAD+1,7             LENGTH ENTERED > 7?
*        BL    DISPLY                  NO - ONLY PROGNAME
*
*        READ STARTUP DATA
*
CM$CALL READ,(RETCODE,IPTAREA,WRLEN,NUMLEFT)
         LA    R1,20
         CLI   NUMLEFT+1,20             RETCODE COULD BE TESTED INSTEAD
         BH    MOVE00
         LH    R1,NUMLEFT              ACTUAL LENGTH TRANSFERED
MOVE00   DS    0H
         BCTR  R1,0                    FOR EX
         EX    R1,MOVE1
         EX    R1,MOVE2                MOVE 1ST TABLE ENTRY
         LA    R6,20(,R6)              INCREMENT POINTER
*
*        DISPLAY STARTUP DATA
*
DISPLY   DS    0H
         LA    R1,L0
         STH   R1,WRLEN
         CM$CALL WRTC,(RETCODE,AREA0,WRLEN)
         ICM   RF,15,RETCODE
         BNZ   END                     NO, TERMINATE PROGRAM
*
WHOISIT  DS    0H
         MVC   IPTAREA,BLANKS          CLEAR INPUT AREA
         CM$CALL READ,(RETCODE,IPTAREA,RDLEN,NUMLEFT)
         ICM   RF,3,NUMLEFT            ANY DATA?
         BZ    END                     NO
         CLC   IPTAREA(6),RECALL       ARE WE RECALLING TABLE ENTRY?
         BE    WHICHONE                YES, BRANCH
         MVC   0(20,R6),IPTAREA        MOVE INPUT TO TABLE
         LA    R6,20(,R6)              INCREMENT POINTER
WENTER   DS    0H
         LA    R1,L1
         STH   R1,WRLEN
*
*        WAIT FOR "TIME" SECONDS FOR OPERATOR REPLY
*
```

```
         CM$CALL WRTC,(RETCODE,AREA1,WRLEN,,TIME)
         ICM   RF,15,RETCODE           IF TIMEOUT TERMINATE
         BNZ   END                     NO, TERMINATE PROGRAM
         CR    R6,R5                   END REACHED?
         BNH   WHOISIT                 NO
         LA    R6,WSTABLE              RESTART FROM BEGIN
         B     WHOISIT                 READ NEXT
WHICHONE DS    0H
         LA    R8,WSTABLE              POINT TABLE START
         OC    IPTAREA+7(2),X2F0       BE SURE IT IS NUMERIC
         PACK  WRKCOUNT,IPTAREA+7(2)
         ZAP   TBLCOUNT,INCRP
         LA    R1,9                    LOOP COUNT
COMPARE  CP    WRKCOUNT,TBLCOUNT
         BE    FOUND
         AP    TBLCOUNT,INCRP
         LA    R8,20(,R8)
         BCT   R1,COMPARE              TRY AGAIN
*
FOUND    MVC   NAME,0(R8)
         LA    R1,L2
         STH   R1,WRLEN
*
*        DISPLAY RECALLED ENTRY FOR "TIME " SECONDS.
*        NOTE THAT KEYBOARD REMAINS LOCKED
*
         CM$CALL WRTR,(RETCODE,AREA2,WRLEN,,TIME)
         ICM   RF,15,RETCODE
         BE    WENTER
*
END      DS    0H
         LA    R1,L3
         STH   R1,WRLEN
*
*        DISPLAY LAST MESSAGE AND TERMINATE THE PROGRAM
*
         CM$CALL WRTD,(RETCODE,AREA3,WRLEN)
*
CANCEL   DS    0H
         MCALL ABEND,ABCODE=0001
*
*---------------------------------------------------------------
*        WORK
*---------------------------------------------------------------
MOVE1    MVC   OUTAREA(0),IPTAREA
MOVE2    MVC   0(0,R6),IPTAREA
DWRD     DS    D
SAVE     DS    18F
RETCODE  DS    F
IPTAREA  DS    CL80
NUMLEFT  DS    H
NUMREAD  DS    H
X2F0     DC    X'F0F0'
WRLEN    DC    H'80'
RDLEN    DC    H'20'
PRGLEN   DC    H'6'
INCRP    DC    PL2'1'
TBLCOUNT DC    PL2'0'
WRKCOUNT DC    PL2'0'
TIME     DC    H'10'
BLANKS   DC    CL20' '
```

```
RECALL   DC    CL6'RECALL'
AREA0    DS    0CL80
         DC    C'PROGRAM NAME= '
PROGNAME DS    CL7
         DC    X'15'
         DC    C'TOTAL LENGTH ENTERED= '
TLEN     DS    CL2
         DC    X'15'
         DC    C'INITIAL DATA= '
OUTAREA  DS    CL20
AREA1    DS    0CL73
         DC    X'15'
         DC    X'15'
         DC    C'ENTER A 20 CHARACTER STRING OR ''RECALL'''
         DC    X'15'
DC    C'AND A 2 POSITION NUMBER FROM 1 - 10'
L0       EQU   *-AREA0
AREA2    DS    0CL80
         DC    X'15'
         DC    X'15'
NAME     DS    CL20
         DC    58C' '
L1       EQU   *-AREA1
L2       EQU   *-AREA2
AREA3    DC    C'PROGRAM TERMINATED NORMALLY'
L3       EQU   *-AREA3
         DS    0F
WSTABLE  DC    9CL20' '
LASTENT  DS    CL20' '
*
         LTORG ,
         COPY  CCREGS
         COPY  CCCOMREG
         END
```

Example 2 - Terminal-Independent I/O using delimiter list

This sample program does basically the same as the previous one, except that more than 1 table entry can be entered at each prompt, separated by commas.

```
COPY  CCGLOBS

*
*        REGISTERS ON ENTRY:
*              R2 = A(COMREG)
*              RD = A(CALLER'S SAVE AREA)
*              RE = RETURN ADDRESS
*              RF = ENTRY POINT
*
SAMP2    CSECT
         USING SAMP2,RC
         STM   RE,RC,12(RD)
         LR    RC,RF                   LOAD ENTRY POINT
         ST    RD,SAVE+4
         LR    R3,R1                   SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
         LR    RD,R1
         LA    R0,IPTAREA
         LA    R6,WSTABLE
         LA    R5,LASTENT
```

```
*
*        WRITE INITIAL MESSAGE
*
DISPLY   DS    0H
         LA    R1,L1
         STH   R1,WRLEN
         CM$CALL WRTC,(RETCODE,AREA1,WRLEN)
         ICM   RF,15,RETCODE
         BNZ   END                    NO, TERMINATE PROGRAM
*
AGAIN    DS    0H
         MVC   IPTAREA,BLANKS         CLEAR INPUT AREA
         CM$CALL READ,(RETCODE,IPTAREA,RDLEN,,DLMLIST)
         CLC   IPTAREA(6),RECALL      ARE WE RECALLING TABLE ENTRY?
         BE    WHICHONE               YES, BRANCH
         ICM   RF,3,DNUMRET           ANYTHING READ?
         BZ    DISPLY                 NO, END OF LIST
         MVC   0(20,R6),IPTAREA       MOVE INPUT TO TABLE
         LA    R6,20(,R6)             INCREMENT POINTER
         CR    R6,R5                  END REACHED?
         BNH   AGAIN                  NO
         LA    R6,WSTABLE             RESTART FROM BEGIN
         B     AGAIN
*
WHICHONE DS    0H
         LA    R8,WSTABLE             POINT TABLE START
         OC    IPTAREA+7(2),X2F0      BE SURE IT IS NUMERIC
         PACK  WRKCOUNT,IPTAREA+7(2)
         ZAP   TBLCOUNT,INCRP
         LA    R1,9                   LOOP COUNT
COMPARE  CP    WRKCOUNT,TBLCOUNT
         BE    FOUND
         AP    TBLCOUNT,INCRP
         LA    R8,20(,R8)
         BCT   R1,COMPARE             TRY AGAIN
*
FOUND    MVC   NAME,0(R8)
         LA    R1,L2
         STH   R1,WRLEN
         CM$CALL WRTC,(RETCODE,AREA2,WRLEN)
         ICM   RF,15,RETCODE
         BZ    DISPLY
*
END      DS    0H
         LA    R1,L3
         STH   R1,WRLEN
*
*        DISPLAY FINAL MESSAGE AND TERMINATE THE PROGRAM
*
         CM$CALL WRTD,(RETCODE,AREA3,WRLEN)
*
CANCEL   DS    0H
         MCALL ABEND,ABCODE=0001
*
*---------------------------------------------------------------
*        WORK
*---------------------------------------------------------------
SAVE     DS    18F
RETCODE  DS    F
IPTAREA  DS    CL160
NUMLEFT  DS    H
X2F0     DC    X'F0F0'
```

```
WRLEN    DC    H’80’
RDLEN    DC    H’20’
PRGLEN   DC    H’6’
INCRP    DC    PL2’1’
TBLCOUNT DC    PL2’0’
WRKCOUNT DC    PL2’0’
TIME     DC    H’10’
BLANKS   DC    CL160’ ’
RECALL   DC    CL6’RECALL’
AREA1    DS    0CL73
         DC    X’15’
         DC    X’15’
         DC    C’ENTER ONE OR MORE TABLE ENTRIES UP TO 20 CHARS LONG ’
         DC    C’SEPARATED BY COMMAS OR’
         DC    X’15’
         DC    C’’’RECALL’’ AND A 2 POSITION NUMBER FROM 1 TO 10’

AREA2    DS    0CL80
         DC    X’15’
         DC    X’15’
NAME     DS    CL20
         DC    58C’ ’
L1       EQU   *-AREA1
L2       EQU   *-AREA2
AREA3    DC    C’PROGRAM TERMINATED NORMALLY’
L3       EQU   *-AREA3
         DS    0F
WSTABLE  DC    9CL20’ ’
LASTENT  DS    CL20’ ’
*
DLMLIST  DS    0F
DLMQUAN  DC    H’1’
DNUMRET  DC    H’0’
DLMNUM   DC    H’0’
DLIMITR  DC    C’,’
*
         LTORG ,
         COPY  CCREGS
         COPY  CCCOMREG
         END
```

Example 3 - Terminal-Dependent Output

This sample coding illustrates the use of a terminal-dependent output with erase option (*WRTSEC*). Note that Buffer addresses must be specified as binary values relative to zero (upper left corner).

```
SCREEN   START
  ...
  ...
  ...
WRTRTN   DS    0H
         CM$CALL WRTSEC,(RETCODE,SCREEN,IOLEN)
  OC    RETCODE,RETCODE
         BNZ   ERROR
  ...
  ...
ERROR    DS    0H
  ...
  ...
SCREEN  DS    0D
  DC    X’C3’                  WCC
```

```
         DC    X’11’                  Set Buffer Address
         DC    AL2(0)                 Relatibe Buffer Address
         DC    X’13’                  INSERT CURSOR
         DC    C’LINE 1’              DATA: LINE 1
         DC    X’11’                  SBA
         DC    AL2(80)                RBA
         DC    C’LINE 2’              DATA: LINE2
IOLEN  DC    H’20’
  ...
  ...
  END
```

Example 4 - Terminal I/O using Map

This example shows the use of macros to create a map for a defined screen layout and it’s usage in terminal I/O functions.

Screen Layout

```
 Name: .....name field.....
 Addr: ....address field.......
 City: ......city field........
 SSN:  999999999        EMP.NR.: 9999.99
 Gross Pay: 9999999

 Enter Desired Action:  ..action..
```

Map definition

Below is an example showing how to define the map using macros. For ease of use, however, it is recommended to create maps using the UMAP Utility.

```
MAP1F2   START
MAP1F2   MAPSTART F2,TCC=KREBF,FDCDEF=R           alphanum., reqd
*
*  The field below is non-modifiable and outside the data area
*
ERRDIS   MAPF  (1,2),30,OFFSET=-40,FDC=S
         MAPF  (2,2),’Name:’,FDC=SDKY                    CONSTANT
NAME     MAPF  ,20,OFFSET=0,A,FDC=UDKOY                  ALPHANUMERIC
         MAPF  (3,2),’Addr:’,FDC=SDKY
ADDR     MAPF  ,24,OFFSET=20,A,FDC=UDKOY
         MAPF  (4,2),’City:’,FDC=SDKY
CITY     MAPF  ,24,OFFSET=44,A,FDC=UDKOY
TSSN     MAPF  (5,2),’SSN:’,FDC=SDKY
SSN      MAPF  ,9,OFFSET=68,TYPE=Z,FDC=UDKOY             ZONED DECIMAL
         MAPF  (5,25),’Emp.Nr.:’,FDC=SDKY
EMPN     MAPF  (05,034),8,OFFSET=77,TYPE=F,FDC=UDKOY   BINARY FULLWORD
         MAPF  (6,2),’Gross Pay:’,FDC=SDKY
GPAY     MAPF  ,7,4,OFFSET=85,TYPE=P,FDC=UDKOY,DECPLAC=2  PACKED 7,2
TENT     MAPF  (9,2),’Enter Desired Action:’,FDC=SDKY
*
REQ      MAPF  ,10,OFFSET=-40,A,FDC=UDKOY
         MAPEND
         END   MAP1F2
```

Note:
The display characteristics of the constant fields labeled "TSSN" and "TENT" can be modified in the program since a name is specified.

Sample Program

```
SAMP4    CSECT
         USING SAMP4,RC
         ...
         ...
         MVC   MRCBNAM,=CL4'MAP1'      MAP MAP1XX
         MVI   MRCBVERS,C'B'           VERSION
         MVI   MRCBFCTF,C'L'           LONG FCT FORMAT
         MVC   MRCBFBAL,=H'30'         LENGTH OF FEEDBACK IS 30
         ...
INIT     DS    0H                      INITIALIZE DATA BUFFER
         MVI   ERROR,C' '
         MVC   ERROR+1(L'ERROR-1),ERROR
         MVC   NAME,ERROR
         MVC   ADDR,ERROR
         MVC   CITY,ERROR
         MVC   REQUEST,ERROR
         MVC   SSN,=CL9'000000000'
         MVC   GPAY,=PL4'0'
         MVC   EMPNUM,=F'0'
         ...
         ...
PROMPT   DS    0H                      RESET OPTIONS
         MVI   MRCBWOPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
         MVI   MRCBROPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
         CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF)
         L     RF,RETCODE
         CH    RF,=H'16'               WAS SCREEN DESTROYED BY MSG?
         BE    PROMPT                  THEN REWRITE THE SCREEN
         LTR   RF,RF
         BNZ   EOJ
         CM$CALL READM,(RETCODE,MRCB,DATABUFF)
         L     RF,RETCODE
         LTR   RF,RF                   OK?
         BZ    GETCOMM                 YES, GET COMMAND
         MVC   ERROR,MRCBFEED          SHOW ERROR
         B     PROMPT                  AND REWRITE SCREEN
*
GETCOMM  DS    0H                      GET COMMAND FROM OPERATOR
         XC    ERROR,ERROR             CLEAR MESSAGE
         LA    R0,COMFCTES             NUMBER OF FCTE'S
         STH   0,MRCBFCTC              SET COUNT
         MVI   MRCBWOPT,C'O'           LETTER O, SAYS O-NLY
         MVI   MRCBROPT,C'O'
         CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF,COMMAND)
         L     RF,RETCODE
         CH    RF,=H'16'               WAS SCREEN DESTROYED BY MSG?
  BE    GETCOMM                 THEN REWRITE THE SCREEN
         LTR   RF,RF
         BNZ   EOJ
         CM$CALL READM,(RETCODE,MRCB,DATABUFF,FCT=COMMAND)
         L     RF,RETCODE
         LTR   RF,RF                   OK?
         BZ    PROCESS                 YES, PROCESS COMMAND
         MVC   ERROR,MRCBFEED          SHOW ERROR
         B     GETCOMM                 AND REWRITE SCREEN
*
PROCESS  DS    0H
         ...
         ...
         MVC   ERROR,=CL30'RECORD SUCCESSFULLY UPDATED'
```

```
         ...
         ...
MESSAGE  DS    0H                DISPLAY MESSAGE
         MVI   MRCBWOPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
         MVI   MRCBROPT,C'A'           A-ALL FIELDS (SAME AS BLANK)
         MVC   MRCBCOUT,=CL6' '        BLANK CURSOR OUT FIELD
         LA    R0,SHOFCTES             NUMBER OF FCTE'S
         STH   R0,MRCBFCTC             SET COUNT
         CM$CALL WRTMC,(RETCODE,MRCB,DATABUFF,SHOONLY)
         L     RF,RETCODE
         CH    RF,=H'16'               WAS SCREEN DESTROYED BY MSG?
         BE    MESSAGE                 THEN REWRITE THE SCREEN
         LTR   RF,RF
         BNZ   EOJ
         B     INIT
*
         ...
         ...
EOJ      DS    0H
         CM$CALL EOJ
*
*---------------------------------------------------------------
*        WORK  AREA
*---------------------------------------------------------------
         ...
         COPY  CCMRCB                  FROM SOURCE LIB
COMMAND  DS    0F
*                    NNNNNN            FIELD NAME
*                          T           TAG
*                           FDC        OVERRIDING FDC
         DC    CL10'ENTER  D  '        CONSTANT   DISPLAY
         DC    CL10'ACTION UD '                   DISPLAY, UNPROTECT
COMFCTES EQU   *-COMMAND/10
*
SHOONLY  DS    0F
         DC    CL10'NAME   P'          NAME       PROTECT
         DC    CL10'ADDR   P'          ADDR       PROTECT
         DC    CL10'CITY   P'          CITY       PROTECT
         DC    CL10'SSN    P'          SSN        PROTECT
         DC    CL10'NUMBER P'          EMPLOYEE   PROTECT
         DC    CL10'GPAY   P'          GROSS PAY  PROTECT
         DC    CL10'ENTER  N'          ENTER      NON-DISPLAY
         DC    CL10'ACTION PD'         ACTION     PROTECT, DISPLAY
SHOFCTES EQU   *-SHOONLY/10
*
DATABUFF DS    0F                      WRTMC, READM DATA BUFFER
ERROR    DC    CL30' '
REQUEST  DC    CL6' '
DRECORD  EQU   *                       RECORD BUFFER
EMPNUM   DC    F'0'
NAME     DC    CL20' '
ADDR     DC    CL24' '
CITY     DC    CL24' '
SSN      DC    ZL9'000000000'
GPAY     DC    PL4'0'
  ...
         ...
*
         LTORG ,
         COPY  CCREGS
         COPY  CCCOMREG
         END
```

Example 5 - LU6.2 TP

This example shows the use of I/O functions in LU6.2 sessions.

```
  COPY  CCGLOBS
*********************************************************************
*
*        SAMPLE PROGRAM FOR LU6.2 SESSION
*
*********************************************************************
*
*        REGISTERS ON ENTRY:
*              R2 = A(COMREG)
*              RD = A(CALLER'S SAVE AREA)
*              RE = RETURN ADDRESS
*              RF = ENTRY POINT
*
*********************************************************************
SAMP5    CSECT
         USING SAMP5,RC
         CMNAME BRANCH=OS
         STM   RE,RC,12(RD)
         LR    RC,RF                   LOAD ENTRY POINT
         ST    RD,SAVE+4
         LR    R3,R1                   SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
  LR     R3,R1                   SAVE A(PARMS)
         LA    R1,SAVE
         ST    R1,8(RD)
         LR    RD,R1
         MVC   PROGNAME,BLANKS
*
*        READ PROGRAM NAME
*        *SAMP5 = 6 CHARACTERS + 1 BLANK = 7
*
         CM$CALL READ,(RETCODE,PROGNAME,PRGLEN)
         CLI   RETCODE+3,4             AMOUNT OF DATA TRANSFERED IS...
         BNH   READ1                   LESS THAN REQUESTED. ERROR
         BAL   R9,CANCEL
*
*        READ 1ST RECORD
*
READ1    DS    0H
         CM$CALL READ,(RETCODE,RECORD1,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    READ2
         BAL   R9,CANCEL
*
*       READ 2ND RECORD FROM CHAIN
*        WRTC WITH LENGTH 0 MUST PRECEDE EACH READ
*

READ2    DS    0H
         CM$CALL WRTC,(RETCODE,RECORD1,ZEROLEN)
         OC    RETCODE,RETCODE
         CM$CALL READ,(RETCODE,RECORD2,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    READ3
         BAL   R9,CANCEL
*
READ3    DS    0H
```

```
         CM$CALL WRTC,(RETCODE,RECORD2,ZEROLEN)
         OC    RETCODE,RETCODE
         CM$CALL READ,(RETCODE,RECORD3,RDLEN,NUMLEFT)
         OC    RETCODE,RETCODE
         BZ    WRITE
         BAL   R9,CANCEL
*
*
*        NOW THE WHOLE CHAIN WAS READ AND WE ARE IN SEND STATE
*
WRITE    DS    0H
         CM$CALL WRTR,(RETCODE,PROGNAME,PRGLEN)
  CM$CALL WRTR,(RETCODE,RECORD2,WRLEN)
         CM$CALL WRTR,(RETCODE,RECORD1,WRLEN)
         CM$CALL WRTD,(RETCODE,RECORD3,WRLEN)
*
CANCEL   DS    0H
         MCALL ABEND,ABCODE=0001
*
*-------------------------------------------------------------
*        WORK
*-------------------------------------------------------------
SAVE     DS    18F
RETCODE  DS    F
PROGNAME DS    CL7
RECORD1  DS    CL25
RECORD2  DS    CL25
RECORD3  DS    CL25
BLANKS   DC    CL25' '
NUMLEFT  DS    H
NUMREAD  DS    H
ZEROLEN  DC    H'0'
WRLEN    DC    H'25'
RDLEN    DC    H'25'
PRGLEN   DC    H'7'
*
         LTORG ,
         COPY  CCREGS
         COPY  CCCOMREG
         END
```

A matching Client TP that may execute anywhere in the network (must run outside Com-plete) should contain an equivalent to the following LU6.2 verbs and options:

VERB	Options
ALLOCATE	Remote Luname: Com-plete APPL MODEname: installation-defined modename TPname: SAMP5 SECURITY: valid Com-plete user ID and password CONVTYPE: MAPPED SYNCLEVEL: NONE or CONFIRM
SEND	AREA: 25 Byte message LENGTH: 25
SEND	same as above
SEND	same as above
RECEIVE	will retrieve "*SAMP5 ", length=7
RECEIVE	will retrieve the 2nd message sent, length=25
RECEIVE	will retrieve the 1st message sent, length=25
RECEIVE	will retrieve the 3rd message sent, length=25 depending on the implementation also CEB (DEALLOCATE)
RECEIVE	will retrieve DEALLOCATE (if not retrieved in above verb)
DEALLOCATE	TYPE=LOCAL (depends on implementation)